WeeBee Code Reference (Undergraduates)

CBP 11-02-22

1. The Story-Writing-Coding API

The table below lists the methods currently available for Actors and Props. The only difference between Actors and Props is that Actors can display emotions. The organization follows a linguistic classification, which stems from the engine's progeny in Story-Writing-Coding. May methods have several versions; e.g., the simplest **jump();** can be called without a parameter, but the jump height can be specified like this **jump(50);**

All methods (actions) take the same time. This is currently set to 2.0 seconds. This can be changed, as explained below. The term "dObj" (dynamic object) refers to either an Actor or a Prop.

Process: Material: Transformative: Enhancing: Motion "move-at"		
Basic Function	*+ param*	
flipH();		horizontal flip
flipV();		vertical flip
spin();	spin(speed);	spin at location
hover();	hover(speed);	same as spin
	spinV(speed);	rotate about vertical axis
jump();	jump(height);	jump
rest();		rest or pause

Process: Material: Creative:		
hide();		hide
show();		un-hide

Process: Material: Transformative: Elaborating: size		
	shrink(scale);	shrink by scale e.g. 0.8
	grow(scale);	grow by scale e.g. 1.2
	squishH(scale);	horizontal scale change
	squishV(scale);	vertical scale change
	squishHV(scaleH,scaleV)	scale change both horizontal and vertical

Process: Material: Transformative: Extending "possession"		
	pickup(dObj);	pick up a Prop or an Actor
	putdown(dObj);	put down a Prop or an Actor

Process: Material: Transformative: Enhancing: motion "move-to"		
	flyto(X,Y);	moves in a straight line to (X,Y)
	flyto(dObj);	moves in a straight line to a Prop or Actor
	walkto(X);	moves horizontally to X, in a straight line
	hopto(X);	makes a parabolic trajectory to X
	runto(X);	moves horizontally to X with a wobble
	flapto(X);	moves in a horizontal line to X with wings flapping (for winged creatures)
	stepto(X);	moves in a horizontal line to X with legs moving (for non-winged creatures)
	leapto(X,Y);	makes a parabolic trajectory to (X,Y)
	leaptoH(X,Y,H);	makes a parabolic trajectory to (X,Y), with control over the leap height
	flyto3D(X,Y)	like flyto(X,Y) but this variant gives a sense of perspective **EXPERIMENTAL**

Process: Mental: cognition		
	thinks(string);	string appears on canvas preceded by actor name
	thinks(string, fontSize);	
Process: Verbal: Projecting		
	says(string);	string appears on canvas preceded by Actor name
	says (string, fontSize);	
	says(string,fontSize,true);	string appears near top-right of actor/prop
	shouts(string);	string appears on canvas without the Actor name
	shouts(string, fontSize);	
	chirps(string)	plays a .wav file, the string is the filename, Cut off after anim time
	chirps(string, true)	same as chirps but plays sound at end of anim time.
	sings(string)	plays a .wav file, the string is the filename. Plays entire file
Process: Relational: Intensive Attributive		
	is(emotion); feels(emotion);	Only for Actors. Changes facial expression.
	appears(image);	Changes the image. Parameter is a string, or a proxy in Header.txt

Process: Existential:		
	add(dObj,X,Y);	adds Actor or Prop at (X,Y)
	add(scenery,X,Y);	adds item of scenery at (X,Y)
	add(scenery,X,Y,front);	*idem* but in front of Actors
Process: Mental: Perception		
	isNear(dObj);	returns boolean true if a dObj is close to another dObj
	canSee(dObj);	returns true if both Props or Actors are on the canvas

Scene Management		
	setScene(sceneID);	Select a built-in scene
	setScene("fname");	Load new scene from image supplied 900 x 600 .jpg
	changeScene("fname");	Load new scene from image supplied 900 x 600 .jpg and remove all previous scenery
showGrid();		Shows the grid on the canvas.
synch();		forces synchronisation of all Props and Actors to this point
tracePaths();		Breadcrumbs dropped when Actors and Props move – shows their paths

Turtle Graphics ("LOGO")		
	moveForward(dist);	Note. These can be accessed by WeeBees and Props.
	rotate(degrees);	
	setpenDown(true_false);	
	setpenDown(true_false, color);	

2. Scenery, Props and Actors

ant

barrel

rush

bush

cloud

dandelion

egg

fire

flower

kite

storm

log

mushroom

robin

rock

rug

saucer

scarecrow

shell

star

sun

sunflower

tree

bigtree

Scenery

Actors

Props

mysaucer	myrobin	myscarecrow
myant	myrock	mystar
myshell	mytree	mymushroom
mybush	myfire	mykite
myrug	mybarrel	myegg
mylog	mysun	

Remember, you can add many instances of each scenery element, but you can only add one instance of an actor and a prop.

3. Standard Programming Constructs

The WeeBee engine is written in Java so you have access to that language, well almost. In the Story-Writing-Coding mode **you cannot use if-then-else** selection statements due to the way the engine works. But the following constructs are useful

3.1 User Functions

Your code will soon extend to tens (or above a hundred) lines, and it is annoying to have to replay all the code when you add a few more lines. So split your code into functions and comment out one function when you are satisfied with the code you have written there. Here's an example where the code in **myFunct1()** is temporarily skipped.

```
public void once() {
  //myFunc1();
  myFunc2();
}
public void myFunc1() {
…
}
public void myFunc2() {
…
}
```

3.2 Loops

Here are two examples of using loops, the first adds multiple scenery elements to the scene and the second makes a character execute multiple actions.

```
int count;

public void once() {
  count = 0;
  while(count < 50){
    add(rock, count,10);
    count += 10;
  }
}
```

```
int count;

public void once() {
  count = 0;
  while(count < 5) {
    grog.jump(10*count);
    count ++;
  }
}
```

3.3 User Input

You can get input from the user by a call to the asks(); function on an actor/prop. In the example below on the left, Grog asks for a size factor which is used as a parameter to the grow(); function.

```
float size;

public void once() {
  add(grog,10,10);
  size = grog.asksForNumFloat("Tell me my new size");
  grog.grow(size);
}
```

```
float height;
public void once() {
  add(grog,50,10);
  height = grog.asksForNumFloat("Input dance height");
  dance(height);
}
public void dance(float h) {
  grog.jump(h);
}
```

The example on the right shows how you would pass an input parameter to your own function. The following asks are available:

grog.asksForNumFloat(…); returns a float

grog.asksForNumInt(…); returns an int

grog.asksForYesNo(…); returns a boolean

4. Configuring the Engine ("Header.txt") and Writing to the console

Here you can set the animation time interval which is currently 2 seconds, **canvas.animationTime=2.0;**

You can also set the text-box font size, **canvas.gui.selectedSourcePanel.setFontSize(12);** or you can choose the font family, style and size, **canvas.gui.selectedSourcePanel.setFont("Ariel",Font.BOLD,18);**

To write to the console you need to add this line to your code, **canvas.gui.msgOutput.setText(string);** Remember a Java string can be built like this, "The current value of count is " + count +" and the jump height is "+ jumpHeight

5. Synchronisation

When you have multiple actors and props in a scene, then it is best to code in "tuples", e.g., with two characters you should code in paired statements, one for each character. Here's the 4 possible tuples for two characters:

pip.rest(); grog.rest();	pip.jump(); grog.rest();	pip.rest(); grog.jump();	pip.jump(); grog.spin();
	"Pip jumps while Grog rests"		"Pip jumps while Grog spins"

So, if you want to code "Pip jumps THEN Grog spins" you would write

```
pip.jump();
grog.rest();

pip.rest();
grog.spin();
```

If you have 3 actors/props then there are 8 possible combination of **rest()** and action in the tuples. There are two other ways of synchronising:

(1) If you use **add(object,X,Y);** in your sequence, then all actors/props previously used will be synchronised after the add.

6. Creating and Instantiating Props and Actors

You can use the normal Java syntax for doing this. Here's an example of how to create a new prop.

```
SceneObject fried;

public void once() {
  fried = new SceneObject(canvas,"friedegg");
  add(fried,20,10);
  fried.jump();
}
```

Here you have created an image with filename "friedegg.png" and have put it in the Data folder. To get an idea of the image size, inspect some images already in the folder. The image type must be **.png**

Creating and using a new Character follows a similar pattern, but you need an image file for each emotion. Say your new character is Izzy, then you need images named something like this Izcontent.png, Izexcited.png and so on for all emotions. Here's how you would use Izzy, the constructor loads all the images for you.

```
WeeBee Izzy;

public void once() {
  Izzy = new WeeBee(canvas,"Iz");
  add(Izzy,20,10);
  Izzy.jump();
}
```

7. Working with Scenes

7.1 Creating your own Background.

This is straightforward. You must create a 900 x 600 pixel **jpg** image and place it in the **data** folder. Let's call this Scene2.jpg You can then set the scene to this, anywhere in your code by this command

```
setScene("Scene2");
```

You can call this function to change scenes on the fly.

7.2 Clearing a Scene

It is possible to clear out all your scenery at any time, though you cannot clear actors or props. If you want to make actors or props disappear from one scene, then reappear at a different place in a new scene then you could do this

```
grog.hide();
grog.flyto(-100,-100);
setScene("newScene");
grog.flyto(30,10);
grog.show();
```

Finally, here is how to clear out all your scenery and load a new background image

```
clearScene(newImage);
```

Unfortunately, there is no way to automatically remove Actors and Props. So relocate them off the canvas.

7.3 Adding your own scenery

Simply use the name of your **.png** file in this variant of add:

add("myScenery",X,Y);

8. Experimental Theatre

8.1 Assets Provided

This is experimental in the sense of coding, not theatre. Instead of an outdoor scene which can support a story, here the animation takes place on a stage. So instead of writing a *story* you could write a *script.* There are three parts to a stage:

(i) The backdrop

(ii) The façade (curtain stuff)

(iii) Flats (vertical surfaces at each side of the stage used to conceal actors due to make an entrance)

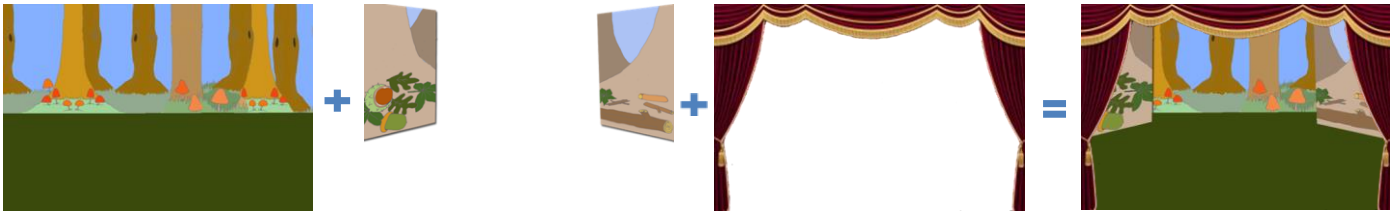
The following code builds a stage with all three components. In addition the flats fly in from the top of the stage. You could build on this code to effect scene changes, by raising and lowering different flats. The façade is added last, since it has to be at the front.

```
setScene(13);  // built-in woodland
scene
add(mywoodflat,45,200);
mywoodflat.flyto(45,0);
add(myfacade,45,0);
```

Here are assets available to create a stage. The names of the flats and façade are already declared and initialised.

Backdrops setScene(N);	**Flats**	**Facades**
11 Bridge	mybridgeflat	myfacade
12 Farmyard	myfarmyardflat	
13 Woodland	mywoodflat	
14 Wintery Scene	mysnowdropflat	

to move, i.e. descend from above, representing scene changes. Here's an example of a backdrop, the associated flat and how they appear combined, and finally with the façade added. Of course additional flats can be



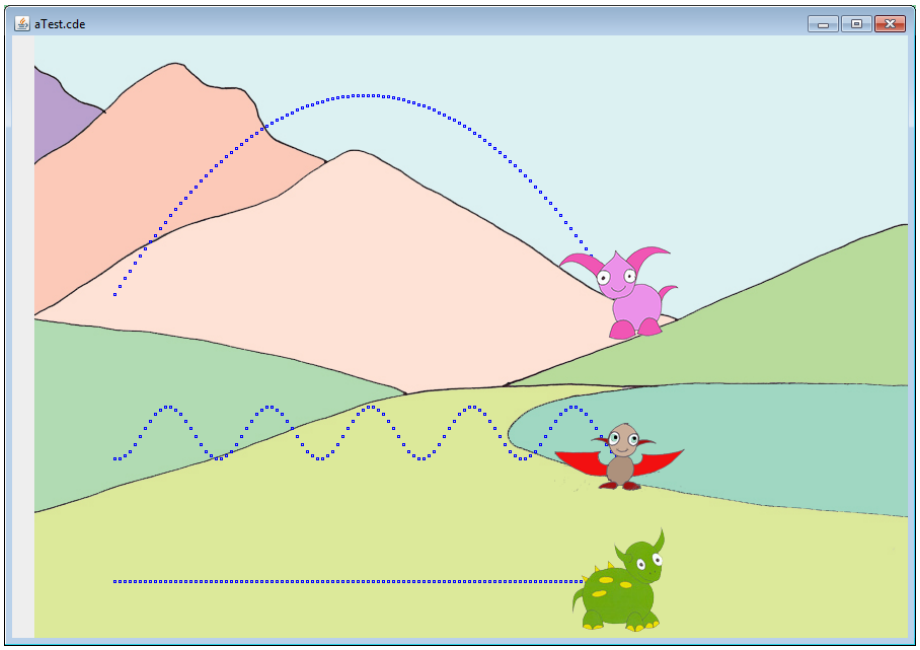
created to provide scene-changes and entire theatrical performances. So far we have not trialled this approach with children.

8.2 Creating your own Theatre Assets

This is straightforward. The backdrop is just a 900 x 600 jpg image, so look at section 6.1 for guidance here. The flats and facades are extended from SceneObject, so think of these as props, and consult section 5.5. Look at the images of the existing flats and facades in the Data folder to see how these have been created.

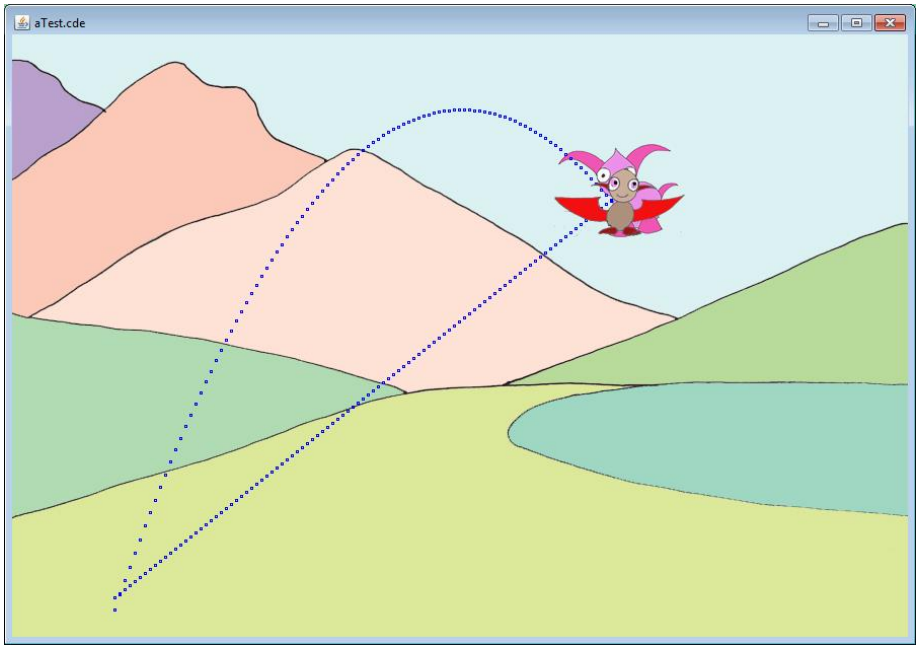
9. The Differences between the "Move-To" methods.

First, let's have a look at the three methods that take a single parameter X. This means they are concerning with movement in the X-direction, so at the end of the movement there is no change in the Y-location of the Actor or Prop. These are shown in the picture below, where **tracePaths();** has been used to create the blue breadcrumbs.



Grog has used the **walkto(X);** method; he moves in a straight line. Flup has used the **runto(X);** method; she bounces up and down as she moves. Pip has used the **hopto(X);** method and her trajectory is the parabola expected of someone moving in gravity. The **flapto(X);** and **stepto(X);** methods behave like **walkto(X);** but either feet or wings move.

Now let's look at the difference between the **flyto(X,Y);** and the **leapto(X,Y)** methods. In the image below, both Pip and Flup have started from (10,0) and have moved to (60,40). Pip who uses the **flyto(X,Y);** method moves along a straight line, from (10,0) to (60,40). Flup, who has used a **leapto(X,Y);** has executed a parabolic arc. But it ends at (60,40). It is a bit like **hopTo(X);** but while the latter will always return the Actor to the ground, you can use **leapto(X,Y);** to jump on top of scenery.



10. Using Sounds

Sound effects, music or spoken narration can be placed in the folder **sounds** using the **.wav** file format. There are two ways sounds can be used:

1) A Long piece of music or narration can be started using e.g., **pip.sings("filename");** The sound will start when this line of code is executed, and will continue for the length of the sound file in seconds. This will therefore accompany the commands which follow.

2) A sound-effect can be played as part of the normal sequence of character actions, e.g., **pip.chirps("filename");** The sound-effect should last no more than 2-seconds. So in the following sequence

```
pip.jump();
pip.chirps("Egg");
pip.spin();
```

Pip will jump, then you will hear the sound of a cracking egg, then Pip will spin.

There is another way of using **chirps();** which will combine a sound with an action. This is shown in the following sequence

```
pip.jump();
pip.chirps("Egg",true);
pip.spin();
```

Here, Pip will jump, then she will spin accompanied by the sound of the cracking egg.

11. Measuring the 'quality' of a story

Some time ago (2016) I reviewed the literature on what makes a good story. A shortened synthesis of my review is shown in the table below. I subsequently used this to evaluate stories and published journal articles based on stories created using the WeeBee engine.

There are 4 categories (A to D) and each category has a number of checks which are weighted (1 – 5) with 5 the best. One number from each category is chosen, and the sum of these is one indication of the overall story quality.

A	**Event Chains (one thing leads to another)**
1	The story has an event chain
1	The chain has an end
3	The chain shows a meaningful causal connections (reason or purpose)
4	The reader can draw *inferences* from events in the chain

B	**Disruption and Restoration of Equilibrium (Story Mountain)**
1	There is a story mountain
2	The story opens by setting the scene
5	There is *surprise* in the story
5	The story creates a sense of *suspense* for the reader

C	**Response of Characters in the story**
4	Characters respond to events
4	Characters show emotional response to events
5	Character behaviour is a consequence of choices they make

D	**The Reader's Experience**
3	The reader's attention was held from beginning to end
5	The reader *empathised* with character(s)
5	The reader was *immersed* in the story
5	The reader was able to fill in any gaps or predict events